A Power Efficient Neural Network Implementation on Heterogeneous FPGA and GPU Devices

Yuexuan Tu, Saad Sadiq, Yudong Tao, Mei-Ling Shyu
Department of Electrical and Computer Engineering
University of Miami
Coral Gables, FL, USA
Emails: $\{yxt120, s.sadiq, yxt128, shyu\}$*@miami.edu*

Shu-Ching Chen
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
Email: chens@cs.fiu.edu

***Abstract*—Deep neural networks (DNNs) have seen tremendous industrial successes in various applications, including image recognition, machine translation, audio processing, etc. However, they require massive amounts of computations and take a lot of time to process. This quickly becomes a problem in mobile and handheld devices where real-time multimedia applications such as face detection, disaster management, and CCTV require lightweight, fast, and effective computing solutions. The objective of this project is to utilize specialized devices such as Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) in a heterogeneous computing environment to accelerate the deep learning computations with the constraints of power efficiency. We investigate an efficient DNN implementation and make use of FPGA for fully-connected layer and GPU for floating-point operations. This requires the deep neural network architecture to be implemented in a model parallelism system where the DNN model is broken down and processed in a distributed fashion. The proposed heterogeneous framework idea is implemented using an Nvidia TX2 GPU and a Xilinx Artix-7 FPGA. Experimental results indicate that the proposed framework can achieve faster computation and much lower power consumption.**

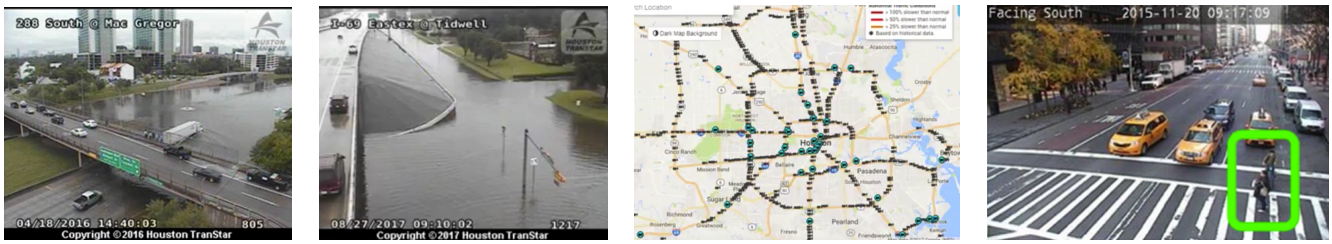
***Keywords*-FPGA, GPU, Heterogeneous Computing, Low-Powered Devices**

I. Introduction

Deep learning has seen many industrial successes over the past decade to many practical problems, including image recognition [8], speech recognition [2] [7] [9], natural language translation [4] [5] [24], etc. By automatically identifying representative features from raw input values, deep neural networks (DNNs) eliminate the time and effort spent on creating hand-crafted features. However, in order to achieve better model performance, it is essential to train the models with large amounts of trainable parameters. For example, BERT [5] showed 5.3% improvements of the validation accuracy when a larger model (340 million parameters, 93.7% validation accuracy) is applied, compared to the baseline model (110 million parameters, 88.4% validation accuracy) for the BLUE benchmark [22]. However, the growth of model size significantly increases the computing time for each sample. Therefore, the performance of deep learning methods is heavily limited in real-time applications where portability, energy efficiency, and accuracy are all equally important. Several multimedia real-time applications such as surveillance camera analysis [16], disaster management [14] [21], person detection [20] are shown in Figure 1. These applications require portable, power-efficient, and computation-efficient hardware to process the data on-site. Heterogeneous computing is considered one of the most promising techniques to meet this scalability demand to handle large deep learning models. Together with CPUs, deep learning computations can be accelerated by graphics processing units (GPUs). However, for distributed edge computing, further acceleration is required to achieve real-time performance and maximize the utilization rate of all computing devices. Model parallelism is considered as one of the best approaches to achieve this goal [3].

Model parallelism can be thought of as partitioning the neural networks into subprocesses, which are computed in different devices. Such parallelism allows a model to be trained distributively and reduces network traffic [3]. This approach is particularly beneficial in big data, multimedia, and/or real-time applications [15] [17] [19] [20] where the size of data inhibits file transfers. In this paper, we propose a model parallelism architecture for DNNs that is distributively computed on low-powered devices. These devices are heterogeneous and consist of a Xilinx Artix-II FPGA and Nvidia TX-2 GPU. The proposed framework accelerates the performance of a simple DNN architecture while reducing its power consumption. This makes model parallelism an ideal fit to compute deep learning models for scalability, real-time, and power efficiency, enabling on-edge real-time services such as surveillance video analysis, speech recognition, and disaster management.

The rest of the paper is organized as follows. Section II describes the related work on heterogeneous computing using FPGAs, GPUs, and CPUs from other publications with different novelties and performance comparisons. The proposed methodology applied in this paper, as well as the framework, are described in section III. The detailed analysis and experimental results are presented in section IV. Finally, the conclusions and future directions are provided.



(a) Disaster scenes detected in surveillance cameras

(b) Map of camera distributes in Houston

(c) Person detection in surveillance cameras

Figure 1: Different real-time multimedia processing applications

Table I: End-to-end comparison among CPU, GPU, and FPGA platforms [25]

Platforms	**CPU**	**GPU**	**CPU+FPGA**
Device	E5-2609	GTX1080	VX 690t
Precision	float	float	fix 16
Frequency (GHz)	1.9	2.1	0.15
Power (Watt)	150	180	26
Latency / image (ms)	733.7	23.5	65.13
Speedup	1x	31.2x	9.7x
Energy Efficiency	1x	26x	65x

II. Related Work

A long-term trend in embedded multimedia applications is the adoption of the FPGA (Field-programmable gate array) as the primary processing engine and supporting devices such as GPUs/ASICs to process floating-point calculations. As depicted in Table I, FPGAs are nearly 7x more power-efficient and relatively faster in inference than GPUs. The heterogeneous environment is programmed using High-Level Synthesis (HLS) and regular programming languages such as OpenCL or C++, allowing for a much higher level of abstraction. However, even with state-of-the-art HLS, programming with FPGAs is still an order of magnitude more difficult. This is because certain components of the neural network can cause bottlenecks if not properly implemented in heterogeneous environments. For example, when using Intels OpenCL compiler, it takes between 4 to 12 hours to compile a typical Convolutional Neural Network (CNN) for the FPGA due to the place-and-route phase.

In model parallelism, batch size, model architecture, and domain parallelism all play important factors in achieving a truly low communication model parallelism [6]. An interesting recent development is to use heterogeneous computing environments (HCEs) in model parallelism. In this case, computation accelerators such as FPGAs [10] can be used to accelerate the model training or inferencing. Thus, assigning each component in a DNN to a proper device becomes a key challenge [12]. [11] [12] proposed reinforcement-learning-based algorithms, where a sequence-to-sequence model is trained in runtime to tune the parallelism scheme and generate the best solution. This shows that the computation time of a DNN model in an HCE with multiple CPUs and GPUs, can be greatly improved if an automated device placement is applied. However, this automated algorithm requires an additional GPU to adjust the scheme in runtime and takes hours to fully converged. Adjusting the placement scheme during runtime can cause large amounts of additional model deployment and communication overheads, and thus makes the algorithm not scalable.

Furthermore, FPGAs have also been incorporated to accelerate CNN computation. Table I shows an end-to-end comparison with existing optimized CPU and GPU solutions for VGG16 [25]. With on-board (VX 690t) testing, 16-bit fixed-point operations demonstrate a 9.7x speedup and 65x energy efficiency over a 4-core CPU (E5-2609), and 2.5x energy efficiency over cuDNN implementations on an Nvidia GTX1080 GPU.

Since FPGAs are more efficient to process binary data with logic gates, [13] proposed to implement Binarized neural networks (BNNs) on FPGAs, and compared its computing time and power efficiency on GPUs. The results indicate that GPUs are better for training but worse at inferencing. As addressed by [23], there is no clear winner among different computing architectures, and each has its own advantages for some specific applications. Meanwhile, [18] proposed an FPGA-GPU-CPU framework that achieves real-time cardiac physiological optical mapping. A real-time locating system based on an HCE using FPGA, GPU, and CPU was proposed in [1]. The platform achieves the best balance among performance, cost, and flexibility by assigning the tasks according to the characteristics of different technologies. In [18], an energy-efficient hybrid architecture was developed to accelerate face recognition using FPGA and GPU. Chip designers also focus on providing HCE platforms. For example, Nvidias Tesla T4 GPU consists of embedded FPGA coupled with ASIC TPUs to accelerate inferencing. Intels Nervava neural network processor platform addresses training and inference separately, using Intel NNP-L 1000 and NNP-I 100 respectively. Thus, it is evident that the general consensus is towards developing efficient HCEs. However, the challenges with low latency data transfer, load balance among devices, limitation of on-

chip memory, threads, etc. inhibit rapid developments in this area.

III. Framework

The objective of this study is to develop deep learning inferencing architectures with FPGAs and GPUs to enable scalable and real-time deep learning for multimedia applications. We achieve this goal by using a combination of FPGA and GPU devices with model parallelism. FPGAs are well suited to perform real-time machine learning and can achieve a deterministic latency of within microseconds. However, FPGA implementations require the hardware circuit design, which is a tedious and costly process. Moreover, when dealing with multimedia data and floating-point operations, the throughput of FPGAs is only one-third of a 75W GPU solution on Nvidia T4. This is because the GPU cores are native hardware for floating-point processing, which can run massive amounts of floating-point operations every clock cycle. This makes GPUs a natural fit for floating-point-intensive applications in real-time signal and image processing. Thus, we use FPGAs as the primary processing engine and GPUs in a supporting role to process floating-point heavy calculations. The end result is a heterogeneous computing architecture that dramatically accelerates training and inferencing to enable a scalable and real-time DNN. The general high-level design of this idea is illustrated in Figure 2. The proposed system mainly contains two computing devices, namely GPU and FPGA. Both devices communicate with each other through a Universal Asynchronous Receiver/Transmitter (UART) serial connection. In the proposed design, processes including convolution, pooling, etc. are realized on the GPU, while fully connected layers are accelerated on the FPGA.

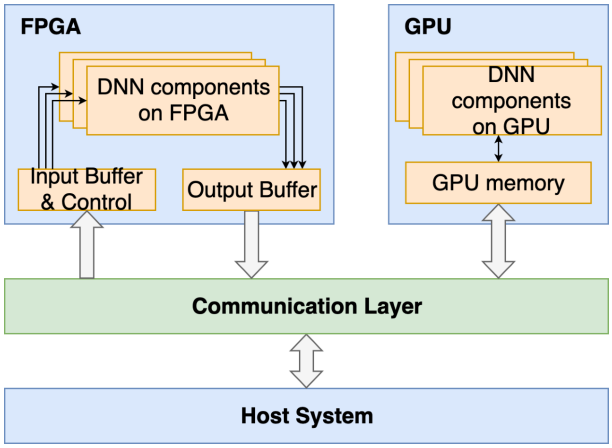


Figure 2: The general device setup of the proposed solution

A. Heterogeneous Framework

The DNN architecture implemented is similar to LeNet-5. We use this architecture to classify large scale hand-written digits in the MNIST dataset. The architecture is primary a CNN. It contains six layers, from which three of them are convolutional layers, two are sub-sampling (max pooling) layers, and the remaining one is a fully connected layer with softmax activation. All convolution layers in the network have 5x5 kernels. Sub-sampling layers are 2x2 max-pooling layers. RELU activation is used throughout the convolution layers in the network.

We break down the DNN architecture by implementing the fully connected layer on the FPGA and the rest on the GPU. This is because GPUs are better suited to perform parallel operations and thus can run convolution operations much faster than FPGAs. Due to the low overhead implementation of FPGAs and the direct realization of the hardware, FPGAs are well suited to accelerate the fully connected layers, which includes a sequential process summing up the weighted inputs. Since this is only a proof of concept, we only implement the fixed-point operations on the FPGA. The complete framework diagram with its modularities is shown in Figure 3.

In order to implement a multi-level representation of the CNN, it is vitally important to understand which of the processes work better on which device. The high number of parameters processed in a DNN can easily overrun the number of slice registers, lookup tables (LUT), and block RAM (BR) of an FPGA. To implement an architecture such as Google's inception-v3 having up to 24 million parameters would require massive control blocks to manage the architecture modules. Moreover, all computations are calculated in real numbers, which requires a resource-intensive logic implementation to handle floating-point operations. Thus, a careful consideration is required when implementing the DNN in a heterogeneous fashion.

The choice of devices for the proposed framework was based on power consumption, rapid prototyping, and modularity advantages. The FPGA used in our proposed framework is an Artix-7 FPGA chip on the Nexys A7-100T development board by Xilinx. This FPGA consists of 15,580 programmable logic slices, each with four 6-input Lookup Tables (LUTs) and eight flip-flops. The GPU employed in the framework is the Nvidia Jetson TX2 embedded computing device. The onboard GPU is a 256-core Nvidia Pascal GPU, while the CPU is a Quad-core ARM Cortex A57 MPCore CPU.

B. Universal Asynchronous Receiver/Transmitter

The proposed algorithm processes the DNN distributively. Thus it is required to connect two devices and establish a reliable interface between the GPU and FPGA. There are several options by which the inter-layer data can be transferred between devices, but we chose to interface the devices using a Universal Asynchronous Receiver/Transmitter (UART).

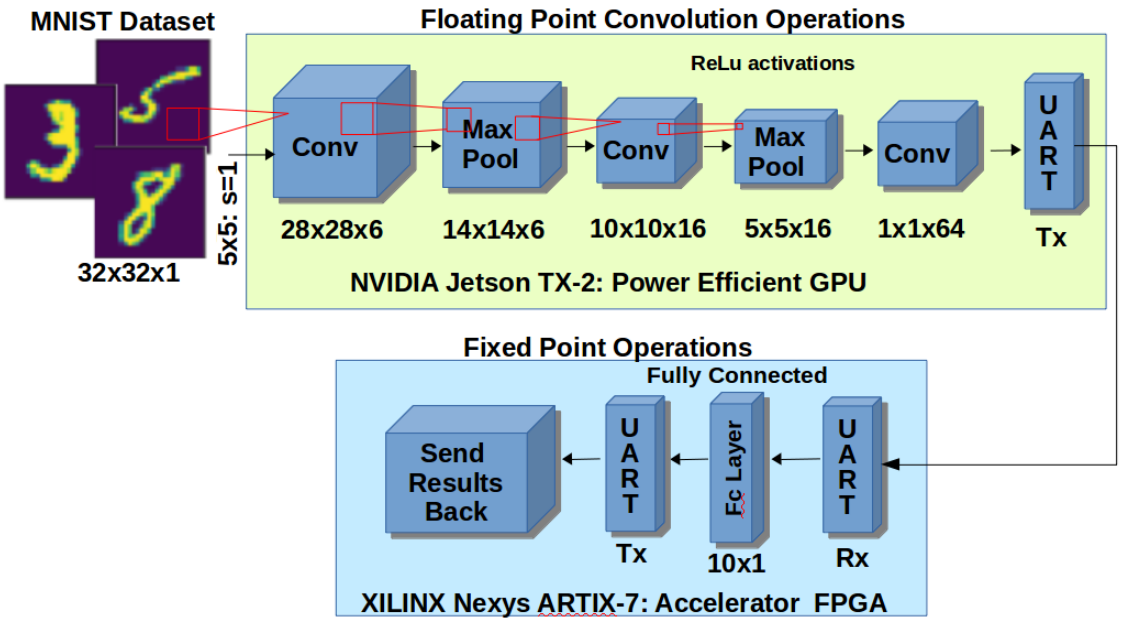


Figure 3: The proposed framework for heterogeneous computing on the FPGA and GPU

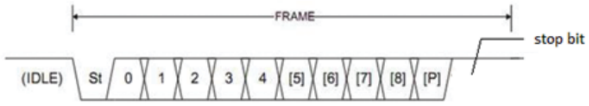


Figure 4: Frame of serially transmitted data

A universal asynchronous receiver is a parallel to serial line converter that sends and receives data in serial and provides the data in parallel to the FPGA. A UART is required when transferring data serially between devices because the data is transferred using Universal Serial Bus (USB). Moreover, the UART handles the mismatch between the voltage levels of the GPU kit and the TTL logic in FPGA (0-5volts). The Nexys A7 development kit has a standard FT2232 HQ USB UART bridge. The FT2232HQ is also used as the controller for the USB-JTAG circuitry. A serial port can transmit 6, 7 or 8 bits at a time with different varieties of start and stop bits as shown in Figure 4. The data transmission starts with logic 0 and terminates with logic 1. In between are the data bits and the parity bit that is optional. The stop bits can be selected from 1, 1.5 or 2 bits. In serial transmission, the transceiver system is first set upon a few parameters that overlay the guidelines of the transmission and reception. These parameters are the number of data bits, parity bit, number of stop bits, and baud rate that describes the speed of the overall data exchange. The baud rate can be selected from 2400, 4800, 9600 or 19200 bauds.

Since a UART system follows a sequence of a finite number of procedures, we implement a finite state machine to control the system data flows. The UART transmitter is similar to the UART receiver, apart from the different flags and control bits. The devices signal a flag high when the entire data has been received by providing an acknowledgment to the program controller. The block diagram of the overall UART system is shown in Figure 5, where the baud rate generator produces the working clock for the system, receiver converts the serial signal rx into parallel data, transmitter performs the data transformation in the reverse way, and both receiver and transmitter are connected to a First-In First-Out (FIFO) to cache the data. The other components in the system will load data from or push data into the corresponding FIFO in order to read to write data from GPU.

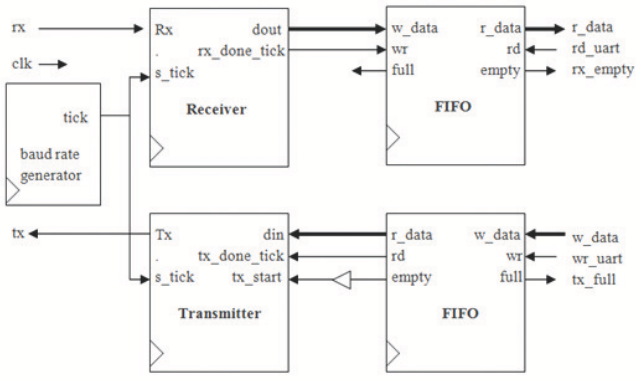


Figure 5: The complete UART system as implemented on the GPU and FPGA

Table II: Nexys A7-100T Brief Technical Specifications

Product Variant	Nexys A7-100T
FPGA Part Number	XC7A100T-1CSG324C
Look-up Tables (LUTs)	63,400
Flip-Flops	126,800
Block RAM	1,188 Kb
DSP Slices	240
Clock Management Tiles	6

Table III: Jetson TX2 Brief Technical Specifications

Product Variant	Jetson TX2 module
GPU	256-core NVIDIA Pascal GPU
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM Cortex-A57 MPCore
Memory	8GB 128-bit LPDDR4 Memory
Storage	32GB eMMC 5.1
Power	7.5W / 15W

The UART system used in the proposed framework is a sub-optimal solution due to its relatively slow speed. Serial interfacing is usually not recommended in the final realization due to its slow speed, high overhead, and the requirement for additional hardware modules. Alternate options like system bus interfaces can be very efficient and may provide optimal overhead times. As a future development task, we aim to use Intel's DE5a-Net DDR4 Altera Arria 10 FPGA device. This device supports PCI Express x8 Edge that can significantly reduce the overhead. Since the goal of this paper is to provide a proof of concept by testing and verifying the heterogeneous implementation, the GPU-FPGA interface utilizes UART serial interfacing modules.

IV. Experiments and Results

The proposed heterogeneous framework is tested using a Nexys A7-100T FPGA from Xilinx and a Jetson TX2 module from NVIDIA. The Nexys A7-100T board, previously known as Nexys 4 DDR, contains a 128MiB DDR2 SDRAM and 1,188 Kbits of fast block RAM. It has 15,850 Programmable logic slices, each with four 6-input LUTs and eight flip-flops. The TX2 board is a low-power (7.5W) embedded computing device, which is built around an NVIDIA Pascal-family GPU and loaded with 8GB of memory and 59.7GB/s of memory bandwidth. The communication between the FPGA and GPU board is achieved through UART with RS232 standard. The baud rate is set to be 19,200 without the parity bit. Tables II and III briefly show some technical specifications of the hardware, respectively.

A simple deep learning model, which contains three convolutional layers and one fully connected layer, is tested with the MNIST dataset. The pre-trained model and its weights are loaded to the FPGA board in the initial stage. During inferencing, the GPU will first forward the inputs to calculate the output vector $X[n]$ of the second last layer, where n represents the length of X and is determined by the number of input nodes of the last fully connected layer.

Table IV: End-to-end performance comparison of different hardware configurations

	GPU	GPU+FPGA+overhead	GPU+FPGA
Precision	float	fix 16	fix 16
Time (ms)	4.019	41.625	3.085
Frequency (fps)	248	24	324
Power (Watt)	6	3.7	

Table V: Performance comparison of different feature sizes

Feature Size n	Time (ms)	Frequency (fps)
16	0.73	1370
32	0.995	1005
64	1.064	940

The generated feature vector $X[n]$ will then be transmitted to the FPGA board through UART. After the calculations of the fully-connected layer are finished on the FPGA, the result will be sent back to the GPU through UART. The calculation precision involved in the GPU is floating-point, however, the intermediate feature vector is converted to fix-point numbers before sending to the FPGA.

Table IV shows the end-to-end performance comparisons among different hardware configurations. The Nvidia TX2 GPU is set in the max-P mode, which optimizes the power efficiency of the board. As shown in Table IV, both TX2 GPU and FPGA board can operate with a relatively tight power constraint. Between the two, the FPGA consumes 1.62x less power. In terms of the time latency, the heterogeneous system is faster when the overhead is not included. The overhead involved in GPU+FPGA module mainly comes from the intermediate UART communication, where the data transfer rate is limited to match the baud rate. In our case with baud rate = 19200, one bit requires approximately $1bit/19200bps = 52\mu s$. This UART transmission overhead will be less obvious in a larger model since the speedup in the FPGA will compensate more for communication delay. A better solution is to replace UART by PCI express, which can greatly increase the transmission speed.

Table V compares the FPGA computing time for a single fully connected layer regarding different feature sizes. When the feature size doubles (i.e., the number of nodes in the fc layer increases), the computing time increases 36% and 7% respectively with n = 16 and 32. As the feature size grows larger, the calculation time is expected to grow only slightly, which proves the scalability of our proposed framework.

V. Conclusion

In this paper, a power-efficient neural network implementation on a heterogeneous computing system with both FPGA and GPU is proposed. Considering the different hardware characteristics of FPGAs and GPUs, we use the FPGA to accelerate a fully connected layer and the GPU for the rest. A small prototype framework is implemented using Xilinx Artix-7 FPGA and Nvidia TX2 GPU, where both

devices are designed for power-efficient computations. The experimental results show the performance improvement of the heterogeneous system over the GPU-only system regarding both computation time and power consumption. In the future, our work can be extended in multiple aspects to experiment and profile the performance of various DNN structures on FPGAs. We plan to test (1) networks with multiple layers, (2) other types of neural network layers, and (3) a more powerful FPGA board (e.g., Intel DE5a-Net). We also plan to apply an alternative data transmission method (e.g., PCI Express) for the FPGA-GPU intermediate communication to effectively reduce its overhead.

REFERENCES

[1] Mohammad Alawieh, Maximilian Kasparek, Norbert Franke, and Jochen Hupfer. A high performance fpga-gpu-cpu platform for a real-time locating system. In *European Signal Processing Conference*, pages 1576–1580. IEEE, 2015.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

[3] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *CoRR*, abs/1802.09941, 2018.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[6] Amir Gholami, Ariful Azad, Peter H. Jin, Kurt Keutzer, and Aydin Buluç. Integrated model, batch, and domain parallelism in training neural networks. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 77–86, 2018.

[7] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[10] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. Deep learning on fpgas: Past, present, and future. *CoRR*, abs/1602.04283, 2016.

[11] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. A hierarchical model for device placement. In *International Conference on Learning Representations*, 2018.

[12] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, pages 2430–2439, 2017.

[13] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic. In *International Conference on Field-Programmable Technology*, pages 77–84. IEEE, 2016.

[14] Samira Pouyanfar, Shu-Ching Chen, and Mei-Ling Shyu. Deep spatio-temporal representation learning for multi-class imbalanced data classification. In *IEEE International Conference on Information Reuse and Integration for Data Science*, pages 386–393, 2018.

[15] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, 51(5):92:1–92:36, 2018.

[16] Samira Pouyanfar, Yudong Tao, Anup Mohan, Haiman Tian, Ahmed S. Kaseb, Kent Gauen, Ryan Dailey, Sarah Aghajanzadeh, Yung-Hsiang Lu, Shu-Ching Chen, and Mei-Ling Shyu. Dynamic sampling in convolutional neural networks for imbalanced data classification. In *IEEE International Conference on Multimedia Information Processing and Retrieval*, pages 112–117, 2018.

[17] Samira Pouyanfar, Yudong Tao, Haiman Tian, Shu-Ching Chen, and Mei-Ling Shyu. Multimodal deep learning based on multiple correspondence analysis for disaster management. *World Wide Web*, pages 1–19, 2018.

[18] Santhosh Kumar Rethinagiri, Oscar Palomar, Javier Arias Moreno, Osman Unsal, and Adrian Cristal. An energy efficient hybrid fpga-gpu based embedded platform to accelerate face recognition application. In *IEEE Symposium in Low-Power and High-Speed Chips*, pages 1–3. IEEE, 2015.

[19] Saad Sadiq and Mei-Ling Shyu. Cascaded propensity matched fraud miner: Detecting anomalies in medicare big data. *Journal of Innovative Technology*, 1(1):51–61, 2019.

[20] Saad Sadiq, Marina Zmieva, Mei-Ling Shyu, and Shu-Ching Chen. Reduced residual nets (red-nets): Low powered adversarial outlier detectors. In *Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration*, pages 436–443. IEEE, 2018.

[21] Haiman Tian, Yudong Tao, Samira Pouyanfar, Shu-Ching Chen, and Mei-Ling Shyu. Multimodal deep representation learning for video classification. *World Wide Web*, 22(3):1325–1341, 2018.

[22] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 353–355, 2018.

[23] Qiang Wu, Yajun Ha, Akash Kumar, Shaobo Luo, Ang Li, and Shihab Mohamed. A heterogeneous platform with gpu and fpga for power efficient high performance computing. In *International Symposium on Integrated Circuits*, pages 220–223. IEEE, 2014.

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[25] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.